Assignment 2
2dv604

Waeel Alhrazy (wa222bq)
Alaa Alwan (aa223fw)

Assignment 2 S19 – Persistency and Event-monitoring

Task one: Research.

Architecture pattern provides solutions to explicit problems, although patterns are not always perfect to a system. Patterns provide the system with number of qualities; thus, it should be chosen carefully to achieve better results and deliver the qualities required for the system.

In the first level-decomposition shows, no pattern was used. Therefore, client- server pattern would be chosen to build up the basic structure of the Gymnast League software management. The pattern would support persistency quality which is one of the main qualities supported by patterns.

1- Persistency

It's a quality attribute of a software that represents the state of the data when it's created, saved into a data storage and retrieve all as well in the same time which means to achieve data integrity and availability.

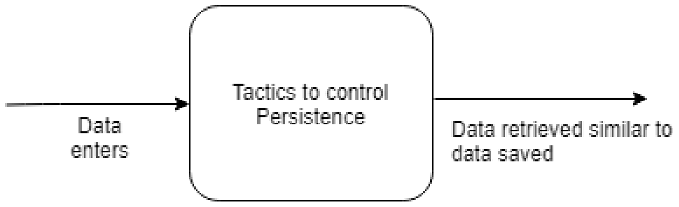
Pattern

Client-Server pattern has its pros and cons, in this system the pros of the chosen pattern is the support of persistency. To support this claim, in case of system shutdown or malfunction this pattern has the ability of data-loss recovery and backup since the whole data stored one place which is the data warehouse. Another advantage of the Client-Server pattern is the improved management of data which leads to minimizing the transaction and data loss. Scalability is also a good quality gained from Client-Server pattern since the server is static with dynamic clients. In addition, scalability is another gained quality from such pattern, since the server is still the same only new clients can be added.

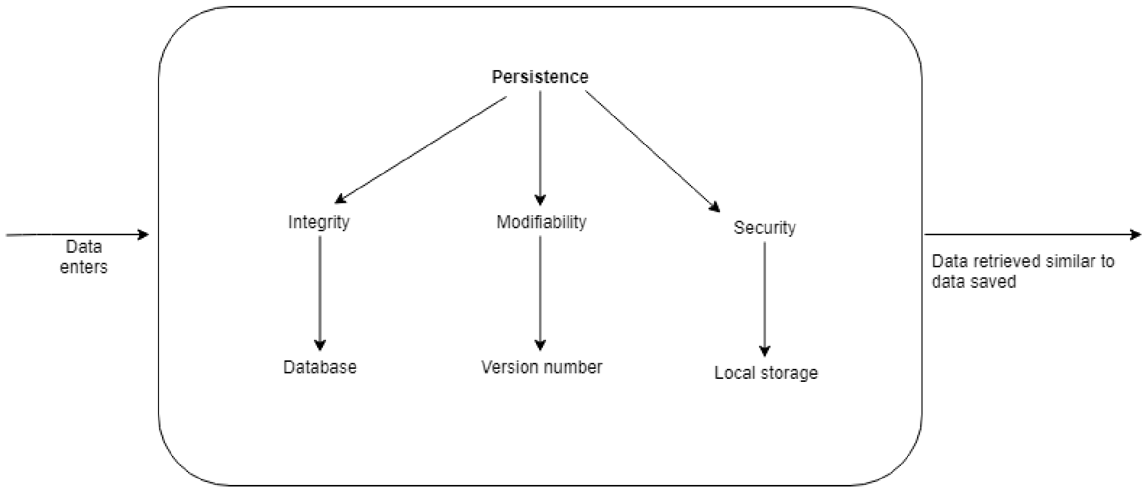
Cons of this pattern is network congestion which affects system performance (quality) due to the solely handling of requests and responses by one server and multiple clients. This could lead to server response/receive failure hence, local storage in client used to avoid data-loss in such case. One more disadvantage could be is Client-Server pattern requires costly management and computing, therefore, to maintain such architecture it would be expensive.

Client-Server pattern is the currently suitable decisions to apply in Gymnastic league management system, another closer option might be Broker pattern which is defined "used to structure distributed software systems with decoupled components that interact by remote services invocations". This pattern is used as a broker(proxy) component between client and server to coordinate the communication such as forwarding requests and transmitting results and expectations. Basically, the broker component receives all the requests from the client which sends the requests to the server as well the server transmit response to broker then from broker to client. In case the server fails, it can be replaced with another server that only connected to the broker component. Without interrupting or informing the client of this change. However, the disadvantage of this pattern is it presents complexity to handle debug and proxies' problems because of the changing environment that surrounds the broker component. Broker pattern increases modifiability and availability of the system; however, it affects the performance and its complexity is an issue in our case.

Tactics


Data enters
Tactics to control Persistence
Data retrieved similar to data saved

The figure above shows the minimum requirement for what we have to accomplish to apply persistency in our architecture. This model will influence our tactics to accomplish persistency in our system. Data in the system should be persistent; available and can cope with modifications on the system or updated version of the data structure.



The second model describes the tactics that we will accomplish to fulfil the requirements and the patterns to implement the tactics. The second model is inspired by the requirements on persistency in model Figure 1.

Data integrity determines when data is created and stored in the data storage without depending on the storage that the data maintain. To deal with different types of data we decide to add a database. However, there is more challenge to it, when data is stored in DBMS, there might be risks that affect persistency like data corruption. To minimize the error rate in transactions, constrains to entities and relation must be applied. Modifiability is another concern that affects persistency, when parts of the system are altered or even the storage structure changed that shows effects on data. In order to solve this problem, whenever storing data altered which affect data handling, when data is stored a version digits will be updated in the record created. Moreover, in case of any problem with handling older data, we can make changes to data handling when inputting old data, different actions to be made to handle it.

Alternative tactics, avoiding data-loss during data transaction is one of the most important characteristics of persistency among the system modules. As the server side receives the data, client can retrieve the data whenever client connected to server side. Thus, the solution is using a local storage in the Setup subsystem to guaranty data availability and minimize data-loss during data transfer. We consider this alternative tactic to be advantageous and proper addition to our system architecture.

2- Event Monitoring

its a process that handles events by collecting, analysing and signalling the occurrences of the events to users. These events occurrences may stem from arbitrary sources in software such as DBMS, application software and processors. Information viewed about individual events or track trends in events to simply identify abnormal behaviour and secure the data transactions. Some examples of referenced architecture for event-monitoring are graphical systems such as X Window System, Microsoft Windows and development tools as SDT.

With event-monitoring tracing data is manageable from system to system and the security level is high, the event structure in this framework depends on client implementation without relying on the server-side support. This makes it an effective pattern to integrate applications without interfering with other existing components. It’s more like client-side design that benefits the existing server-side component instead of designing new server for the system.

Pros of using this framework, when a user enters data using one application, that data should view in other applications used in the same environment. in addition, the user should be able to access all data with a single user experience.

Pattern

Event-bus pattern is a suitable pattern to manage events, it deals with events through four components; event source, event listener, channel and event bus.

Source: publish messages to specific channels on an event bus.

Listeners: subscribe to specific channels and notified of messages published to a channel subscribed before.

Channel: a mechanism to distribute messages.

Event bus: carry messages between source and Listeners.

In this pattern components may subscribe to a set of events and interact with each other via events. The main function of this pattern is to monitor the flow of events and deliver it to all subscribers of that event. Hence, the main form of connecter is event bus, components announce events then connectors deliver those events to the components who registered an interest in those events. Advantage of this pattern its flexibility in dealing with events, new publisher, subscribers and connection can be added easily. As well it is very effective with distributed systems from client-side. Another advantage is components could choose either to react in response of receipt of an event or may ignore it. However, scalability is an issue because of the messages travel at the same time. Another disadvantage that there is no coordination of components into publishers and subscribers which means all components emit and receive events.

Tactics

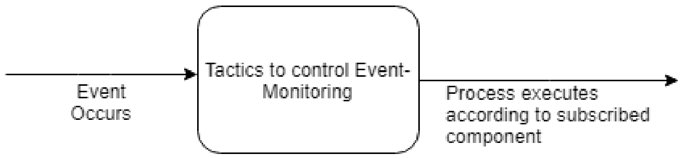
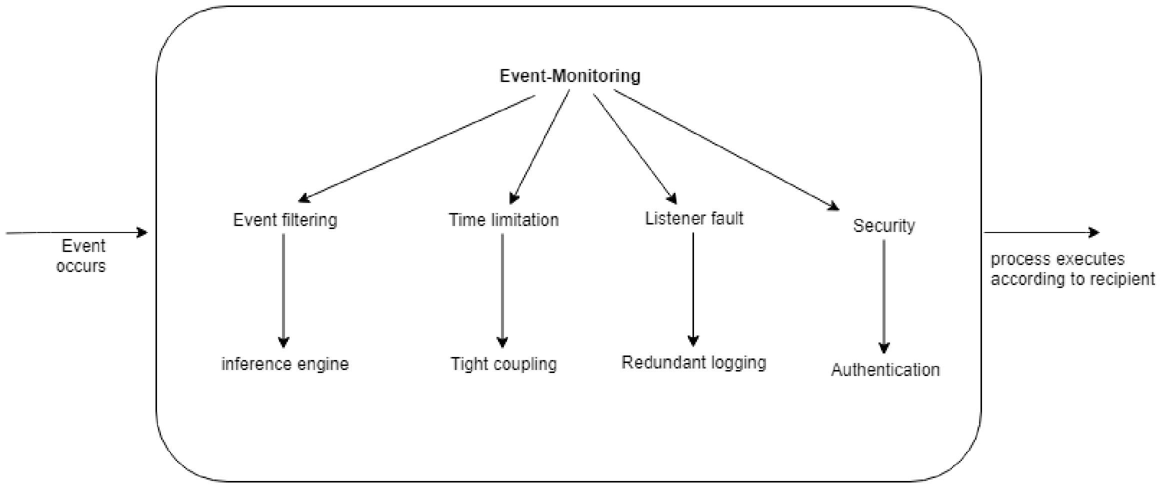

Event
Occurs
Tactics to control Event-Monitoring
Process executes according to subscribed component

Fig above shows the minimum requirement for what we want to accomplish in terms of event-monitoring in our architecture. This model will influence our tactics to accomplish event-monitoring in our system.



First concern is event filtering or collecting, when the event occurs its processed to be filtered in event for analysis to check if the event should be signalled or to occur over the event bus or removed. To solve the problem of detect events or we could say messages, we could use Message inference which is responsible interpreting the message and decide to remove it or keep it. Second delivery time, the broker (server) in Event-bus system may be designed to deliver message these process in the system maybe restricted with time limit to deliver messages but then stop attempting. Whether or not the subscriber confirmed receipt of the messages. To solve this issue a tighter coupling of the design of Event-bus broker must be enforced outside the architecture to assure delivery for instance requiring the subscriber to publish receipt messages. Third concern listener fault, a publisher in Event-bus system might think that the subscriber is listening when its actually not. This will cause a problem because if the event delivers error messages to handle a failure log and the publisher didn't not receive notice of the logger failure. To solve this problem, we can add redundant logging subscribers to the system that are duplicate to the existing logger to increase logging reliability.

Alternative concern is security, a subscriber might be able to receive data that is not authorized which may present a damaging message into the Event-bus system. We can implement an authentication that allow data transaction only if its authorized in the system.

Task two: Select

We select persistency concern to work on our assignment. After completing our research and dig deep into both concerns Event-Monitoring and persistency, we decided to go with persistency.

Task three: The quality (concern) for this particular application/context (Gymnastics):

Aspects of persistency:
According to our decision, software architecture is Server-Client. As result, to achieve the persistency, the application should guaranty two aspects of the persistency.

- **Data persistency**

The server should manipulate the inputs from many users to get the result. That means, all inputs and deferent information will be sent to the server. The data does not belong to any one of the clients. Therefore, manipulate persistent data that is accessed by multiple independent clients. The problem arises in case the server crashed down accidentally. In such case, the last state and all data will be lost, and the users should to re-enter the inputs.

In most cases, users will not be able to provide the information again. Therefore, if the server does not guaranty the data persistency, some inputs will be lost forever. As a result, and to guaranty data persistency, the server should save the last state and data into a data storage. Then the server could restore and acquire that data beyond crash or shut down. By such mechanism, the application will achieve data availability and integrity (i.e. data persistency).

To achieve the previous mechanism, we need to add the component “Data warehouse” which is the mean storage of the system. All data from administrators (such as teams, members, rules…etc.) and scores from judges will be stored into “Data warehouse”. In addition to the last state of the server.

For the stored data, it is divided into three types:

- Raw data: gymnasts, teams, judges, contests, meets, and scores.
- Summary data: schedule, seasonal standings.
- Metadata: a description for the meaning of the teams, judges, contests, meets, and scores.

- **Availability persistency:**

The software architecture is Server-Client. I.e. the end user will enter the new data in the client side, then the client will send data to the server. For Gymnastics, it is important the judges have the ability to use the client application even if the connection between the client and the server was lost.

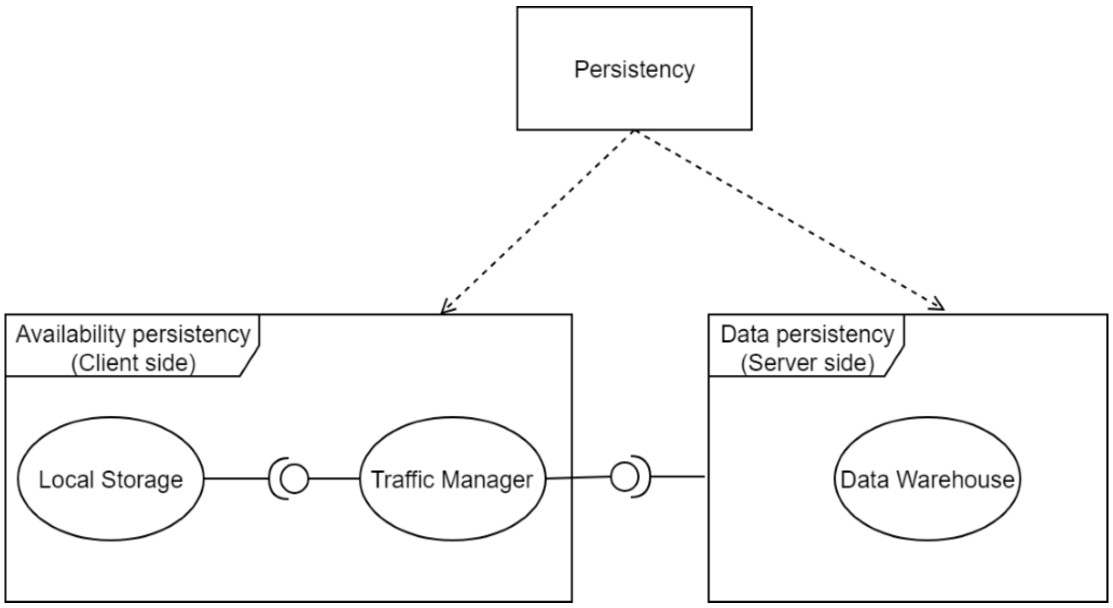
For such case, a local storage should be added to every client is connected to the system. In case the connection is lost, all new data should be saved in the local storage as a backup.

The loss of the connection could be a result for many scenarios. For example, the server itself is powered down, or the network equipment has failed. In such cases, the client will save the data in the local storage until the connection back again and the client can reach the server and send the stored data.

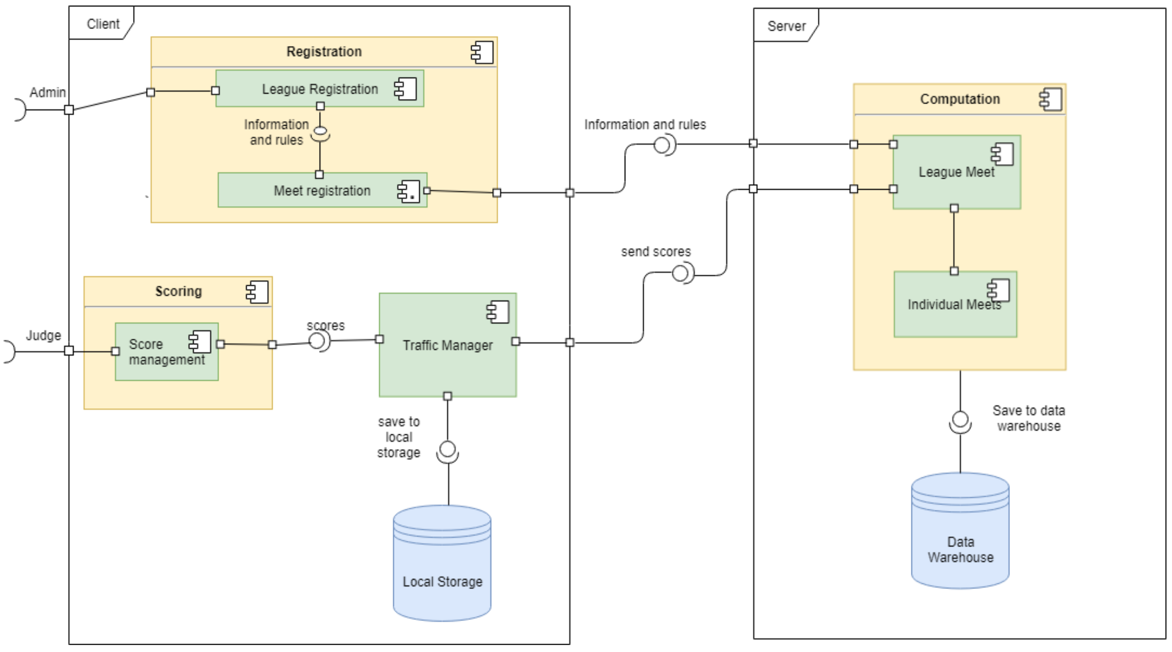
To avoiding send duplicated data or running out of the storage, the client should clear the data from the local storage if they are sent successfully to the server.

To achieve connection persistency, another component “Traffic manager” should be added to the client. The main responsibility for “Traffic manager” is detecting lost connection and switching between the server and the local storage (i.e. switching between online and offline mode). In case the connection was lost, the “Traffic manager” will detect the problem and send the data to the local storage instead of the server to avoiding the loss of data. “Traffic manager” will continuously check the availability of the server by sending periodically a heartbeats or pings request. In case the connection back again, “Traffic manager” will be sending the data (locally stored data) to the server. Then, data will be deleted locally.

Task four: 1st level decomposition for the concern


Persistency
Availability persistency (Client side)
Local Storage
Traffic Manager
Data persistency (Server side)
Data Warehouse

Task five and six: Analyse the architecture and refine



In the figure above we review our first level decomposition after combining it with the persistence concern first level features. As we explained earlier, we choose the Client-Server pattern to build our software and divided our components accordingly.

Our new software architecture designed with Client-Server pattern, as we have added three components to apply the data persistence approach and we explain as follow.

Client side:

- Registration subsystem: admin insert all information about teams, gymnasts and judges. It also manages the gymnast and meet. *Information and rules* interface transfer the inputs to *Meet registration* component, where it manages the rules of teams, meets, judges, and scores then send them to the computation subsystem in server side. This subsystem is responsible for handling the personal data of gymnasts and judges and setting the rules to start the season league.
- Scoring subsystem: actor here is judge where they will input the scores of every event after it ends and add the needed information for instance ratings. *Score management* responsible for keeping and counting the score of each competitor, team, competition and meet all type of scores. *Score* interface will transfer data from and to *Traffic Manager.*
- Traffic Manager: which is responsible of redirecting data between Local storage and *Data warehouse* in server side. It detects lost connection and switching between the server and the local storage (i.e. switching between online and offline mode). In case the connection was lost, the "Traffic manager" will detect the problem and send the data to the local storage instead of the server to avoiding the loss of data.
- Local Storage: responsible of storing data received from traffic manager to save it from getting lost.

Server side

- Computation subsystem: is responsible for collecting all the information and rules from client side then handle the arranging, organising and tracking the meets for instance, which equipment will be used, event location and the meet rotation. *League meet* component responsible for teams over one individual. *Individual meet* responsible for individual competitions. It is also responsible for getting scores from judges input for every event and ratings as well to view them later on. All the data of meets and scores will send to Data warehouse.
- Data warehouse: responsible for saving all types of data from client side and Computation subsystem as well. It also responsible for record keeping for the schedule of meets, assigned judges to the events and seasonal standings. Besides, it saves the score for every competitor.

Integration and design motivation (Decisions)

As we explained previously about selecting data persistency quality to improve our first level decomposition and the architecture reasoning to add three components in our Gymnastic League management system software architecture. We added those components to guarantee data availability as we mentioned in detail in task three. Herewith, we would like to elaborate how we integrated those new components of data persistency quality with our first level decomposition and what decision we made. As we explained in task one the reasons, we chose the Client-Server pattern, it has the ability of data-loss recovery and backup since the whole data stored one place which is the data warehouse serer side.

In our first assignment we had three subsystem Registration, Scoring and Computation subsystems connected through interfaces. It was created with low coupling to be flexible to update or replaced in the future taking into consideration state-of-art-technology to be involved. In this assignment we took the same concept in mind to achieve data persistency. First, we decided upon quality attributes to organize our system using client and server pattern. Registration and Scoring subsystem are made in client side as we think those subsystems would have external interaction (actors) involved to input data for instance Admin for registration and Judge for scores. This distribution of subsystems will enhance the system performance, availability and reliability. We took in consideration connection failure as common worst scenario that could face our system and lead to crash. As we searched and find in some reference architecture for data persistency, it was an effective approach to add Traffic manager component that control the data transfer between client and server. We added local storage to save data as a backup in case the connection to server is down which support data availability and save it from being lost till connection is back.

The server side is designed to handle multiple clients and made Computation subsystem and data warehouse to be part of the server for security reasons. To save our stored data from being manipulated by any user is one reason. Server communicate with server through traffic manager, data is received to compute for meets and events and save the schedules in data warehouse for records keeping and to share upon request from clients. We had a critical decision to take, concerning the Registration subsystem to send rules and information to Server, if it sends data through traffic manager and the connection fails the data will stay in local storage and that would affect the season league to go on. Therefore, in this case we trade-off reliability of data-loss with availability to make sure that information and rules reaches server side.

We would like to introduce some scenarios that could introduce our new components to provide data persistence.

Scenario 1

Target: Local storage

Scenario description: connection between server and local storage failed.

Scenario review: client connected to server of Gymnastic League management software. Its successfully connected and transmitting data to the server. Suddenly the connection died between the client and server due to unexpected reason. The traffic manager task is to check if the client received a response from the server after the data was sent.

Goal*:* the data sent from client must be stored temporarily in someplace.

In the beginning traffic manager observes there is no response received from the server. The traffic manager first attempt to ping the server, if there is no response given then traffic manager redirects the data from client to the local storage. While redirecting the data, traffic manager continues sending pings to the server until the connection is back, then the data will be sent from local storage to the server.

Scenario 2

Target: Traffic Manager

Scenario description: connection between server and local storage failed and traffic manager fails as well.

Scenario review: client connected to server of Gymnastic League management software. Its successfully connected and transmitting data to the server. Suddenly the connection died between the client and server due to unexpected reason. The traffic manager task is to check if the client received a response from the server after the data was sent. However, the traffic manager failed to redirect the data.

Goal*:* to minimize the damage to the quality of the system some action should occur, exactly when the backup for connection failure occurs.

Once the connections fail, between client and server, the traffic manager couldn't redirect the data for some reason, this might lead to data get lost relying on which protocol used.

Scenario 3

Target: Data warehouse

Scenario description: introducing a new structure for storing data.

Scenario review: The data warehouse could store data in different ways depending on the way raw data is received from the client is structured. One possible way is turning the data to decimal numbers.

Goal*:* divide different types of data while storing it in the data warehouse.

New structure for storing data has developed. The previous data records structured with column includes an id type, another id type will be added for the new type of data. Thus, this is taken into consideration when handling the data in the future.

Task seven: Architecture Documentation

7 - Create a view

	Options	**Decision**	**Rationale**	**Evaluation**

Software architecture	Client-Server	Used	**Advantages:** - Good where a set of clients need a set of services. - Better control over the system. - Easier to update resources. - Scalability to increase client. More clients mean more connection to the server and no effect on other clients. - Maintainability. Mostly, maintaining the server is enough. **Disadvantages:** - All resources are available on the server. - Traffic congestion in case an increasing number of connected clients. This will affect performance. - If the connection failed, the clients will lose access to the service.	This pattern improves persistency by controlling of data flaw. In addition to the ability to add a new user to the system without affecting the cost or complexity. The disadvantage “connectionless” could be avoided by allowing clients to work offline (working on the local storage). For the disadvantage “Traffic congestion”, such systems are not exposed to this since the number of clients is limited.
Software architecture	Peer-to-Peer	Not used	**Advantages:** - Decentralized computing. - Scalability in terms of resources and computing power. - Resources are shared among peers. - No system administration since users are admins of their machine. - Less cost for installing new peers and maintaining the connection. - Loss of a peer does not affect other users. **Disadvantages:**	The distributed data is problematic in our case since the data should be persistent. This pattern is more reliable than pattern client-server. But the reliability could be achieved also in the pattern client-server by allowing clients to work offline (working on the local storage). Generally, the features of the pattern Peer-to-Peer do not serve our system.

			- Peers cooperate voluntarily. Therefore, no guarantee for the quality of service. - No guaranteed for security because of increased complexity if too many peers are connected. - Performance depends on the number of peers. - It is difficult to backup data since the data is distributed among all peers.	
Software architecture	Broker pattern	Not Used	**Advantages:** - Allow dynamic change, addition, deletion and relocation of objects. - Distribution is transparent to the developer. - coordinating communication. **Disadvantages:** - Requires standardization of service descriptions. - An additional component should be implemented.	Broker pattern is suitable for systems that have many servers. Our system has a limited number of clients. Therefore, our system has only one server and no need for the broker.
Data persistence	Shared-data Pattern	Used	**Advantages:** - systems store and manipulate persistent data that is accessed by multiple independent components - Shared access to data and support data persistence. - Manage concurrent access to data. - Provide fault tolerance. - Support access control and handle the distribution and caching of data values. **Disadvantages:**	In our system, data produced from one component is needed by other components. In addition, that this pattern allows clients to have concurrent access to data, and this is required in our system since many clients will use the system simultaneously. Moreover, this pattern fulfills the tactic “Data persistency”. The disadvantage of this

			- Maybe a performance bottleneck. - Single point of failure. - Producers and consumers of the shared data may be tightly coupled, through their knowledge of the structure of the shared data.	pattern could be avoided by allowing clients to work offline (working on the local storage).